CS 553 Programming Assignment#1 Benchmarking

This document provides the performance evaluation of CPU Benchmarking, Disk Benchmarking and Memory Benchmarking on Amazon EC2 cloud instance.

Below is the description of the experimental environment of Amazon EC2 cloud. Then the specification of each benchmark is presented with experimental analysis and results in form of table and graph.

1 EXPERIMENTAL ENVIRONMENT

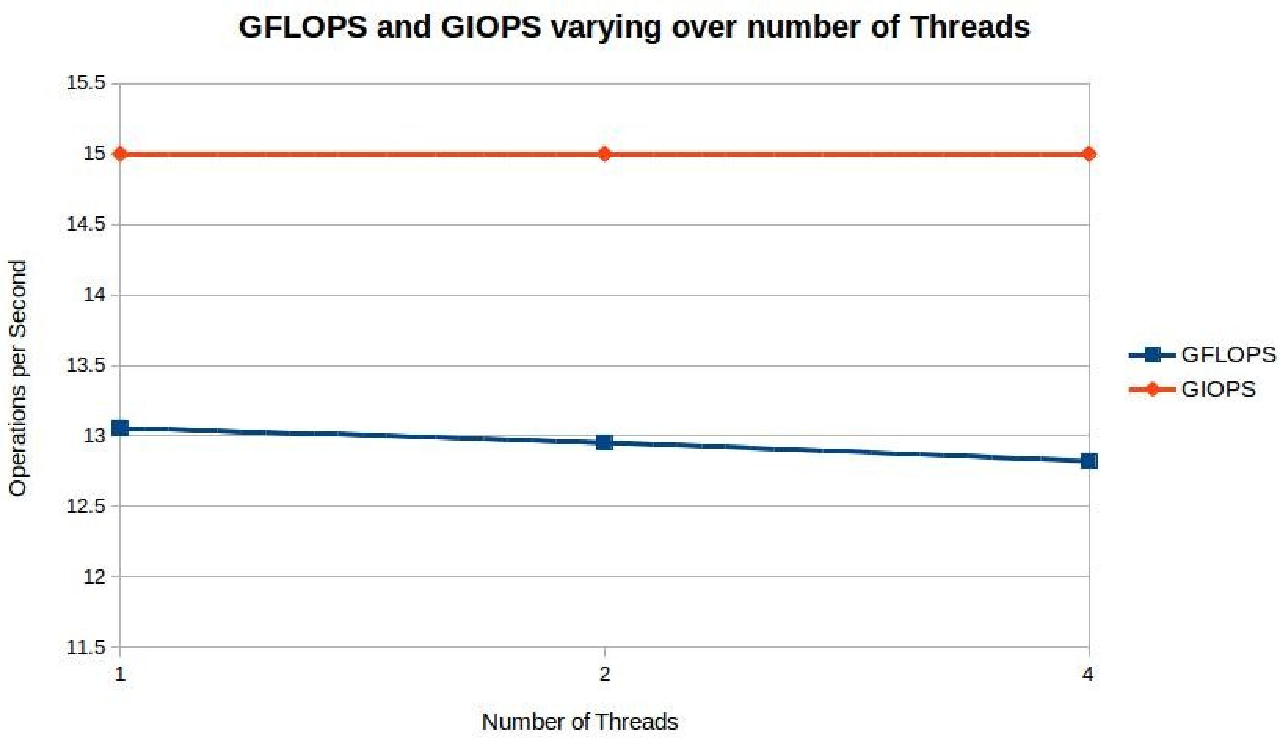
We would be running our experiments of on Amazon EC2 T2 cloud instance. Below screen-shot provides the details of our Experimental Environment.

```
[ec2-user@ip-172-31-27-44 ~]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
Stepping:              2
CPU MHz:               2400.094
BogoMIPS:              4800.18
Hypervisor vendor:     Xen
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0
[ec2-user@ip-172-31-27-44 ~]$
```

2 CPU BENCHMARKING

For CPU Benchmarking I am programmatically finding the GFLOPS (Giga Floating point operations per second) and GIOPS (Giga Integer operations per second) for varying concurrency level (1, 2, 4 threads).

Graph of GFLOPS and GIOPS varying over number of Threads.



No. of Threads	*GFLOPS (Avg.)*	*GFLOPS (s.dev)*	*GIOPS (Avg.)*	*GIOPS (s.dev)*
1B	13.05483	0.00559	15	0.09327
1KB	12.953365	0.02456	15	0.00382
1MB	12.82051	0.03479	15	0.01304

Practical Performance:

FLOPS: 13054830000 GFLOPS: 13.05483

IOPS: 15665796000 IOPS: 15

Theoretical Performance:of AWS-EC2 cloud:

Performance = 1*2.4*16 = 38.4 GFLOPS

Efficiency of CPU Speed (GFLOPS) (33.98 %)

Efficiency = (13.05)/(38.4)

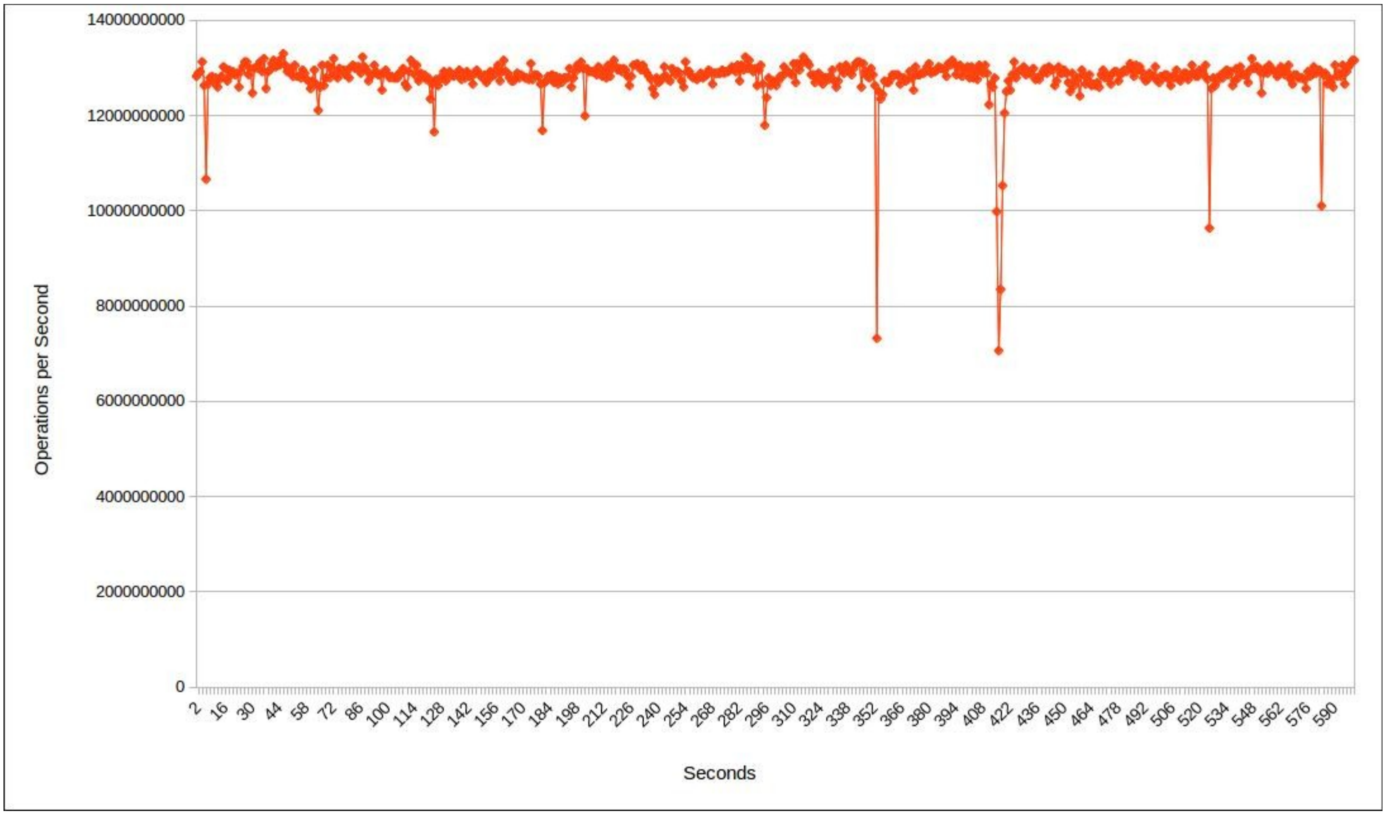
= 33.98 %

Below fig. shows the performance of CPU benchmarking calculated using “Linpack”.

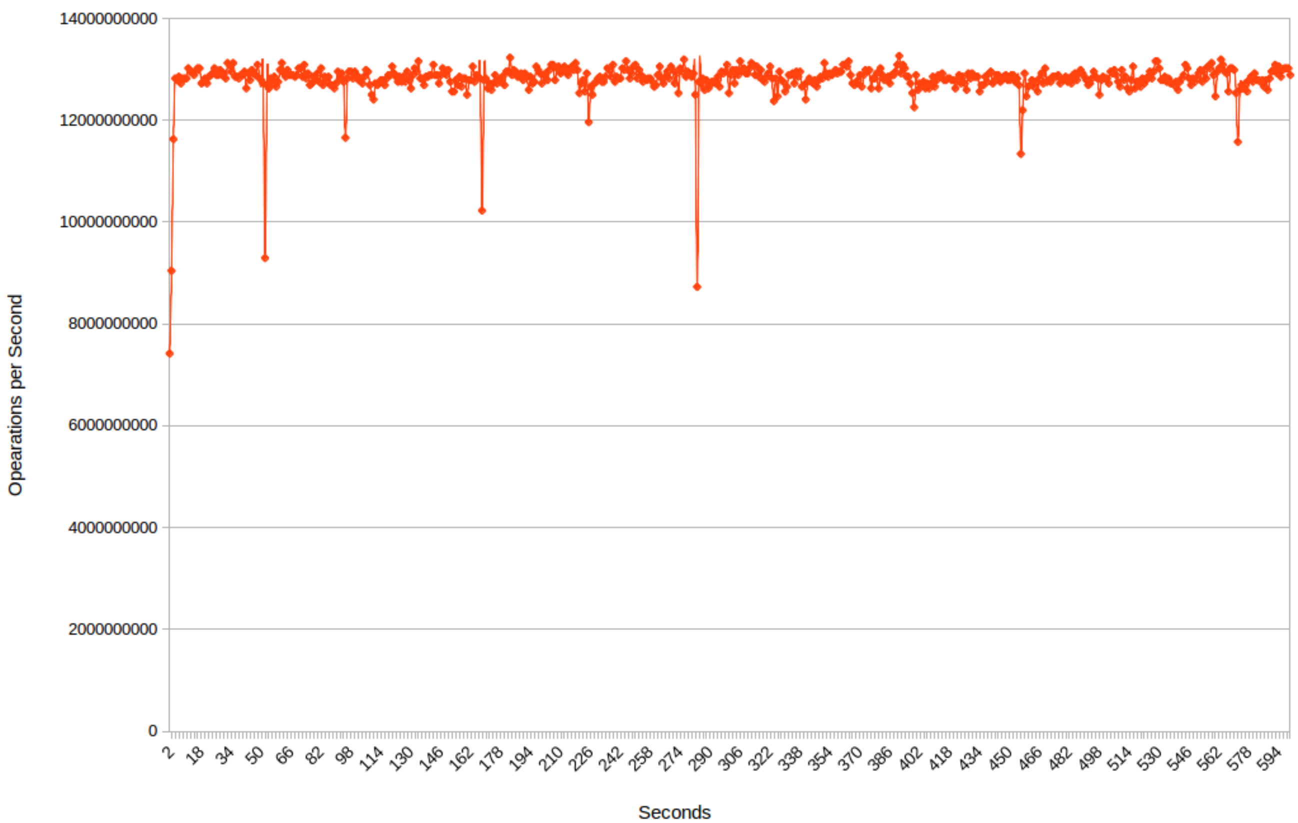
Efficiency of Linpack result to theoretical result = (35.93/38.4) = 93.56%

```
[ec2-user@ip-172-31-49-170 linpack]$ ./runme_xeon64
This is a SAMPLE run script for SMP LINPACK. Change it to reflect
the correct number of CPUs/threads, problem input files, etc..
Fri Feb 12 00:19:36 UTC 2016
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Fri Feb 12 00:19:36 2016

CPU frequency:    2.992 GHz
Number of CPUs: 1
Number of cores: 1
Number of threads: 1

Parameters are set to:

Number of tests: 15
Number of equations to solve (problem size) : 1000  2000  5000  10000 15000 18000 20000 22000 25000 26000 27000 30000 35000 40000 45000
Leading dimension of array                  : 1000  2000  5008  10000 15000 18008 20016 22008 25000 26000 27000 30000 35000 40000 45000
Number of trials to run                     : 4     2     2     2     2     2     2     2     2     2     1     1     1     1     1
Data alignment value (in Kbytes)            : 4     4     4     4     4     4     4     4     4     4     4     1     1     1     1

Maximum memory requested that can be used=800204096, at the size=10000

=================== Timing linear equation system solver ===================

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
1000   1000   4      0.026      26.1378  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      26.2410  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      26.4202  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.026      26.0428  9.632295e-13 3.284860e-02   pass
2000   2000   4      0.195      27.3354  4.746648e-12 4.129002e-02   pass
2000   2000   4      0.193      27.7365  4.746648e-12 4.129002e-02   pass
5000   5008   4      2.494      33.4359  2.651185e-11 3.696863e-02   pass
5000   5008   4      2.489      33.5038  2.651185e-11 3.696863e-02   pass
10000  10000  4      18.682     35.6959  9.014595e-11 3.178637e-02   pass
10000  10000  4      18.440     36.1648  9.014595e-11 3.178637e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       26.2104  26.4202
2000   2000   4       27.5359  27.7365
5000   5008   4       33.4698  33.5038
10000  10000  4       35.9303  36.1648

Residual checks PASSED

End of tests

Done: Fri Feb 12 00:20:25 UTC 2016
[ec2-user@ip-172-31-49-170 linpack]$
```

Below Graph shows FLOPS samples for 600 seconds (10 minutes)



Below Graph shows IOPS samples for 600 seconds (10 minutes)



Result Interpretation:

CPU performance in terms of GFLOPS and GIOPS is almost stable when performing millions of operations as visible through the plotted graphs and tables. The graphs showing FLOPS and IOPS samples taken each second for 10 minutes (600 samples) quantifies our observation that the GFLOPS and GIOPS remain almost stable with some exceptions.

Issues Encountered:

While programming for CPU benchmarking the first problem where I was stuck was the formula to calculate FLOPS and how to find the IPC of a processor which was required to calculate the peek performance. Another problem was whether I need to consider the Wall Clock time or the CPU time while calculating the runtime of the benchmark program.

3 DISK BENCHMARKING

For Disk Benchmarking read and write operations are performed for varying block sizes of 1B, 1KB and 1MB and varying number of threads (1 or 2). The read and write operations are performed sequentially as well randomly with various combination of block size and number of threads. eg. Sequential Read for block size 1B and 1Thread, 1KB block size and 2 Thread and so on. Below are the results of the experiments that I conducted.

Disk Performance: Throughput

Sequential One Thread

Block Size	*Read*	*Write*
1B	10	0.84
1KB	1969.23	860.504
1MB	2621.44	1497.96

Sequential Two Thread

Block Size	*Read*	*Write*
1B	7.69	1.2
1KB	2178.72	966.03
1MB	2097.15	1497.9

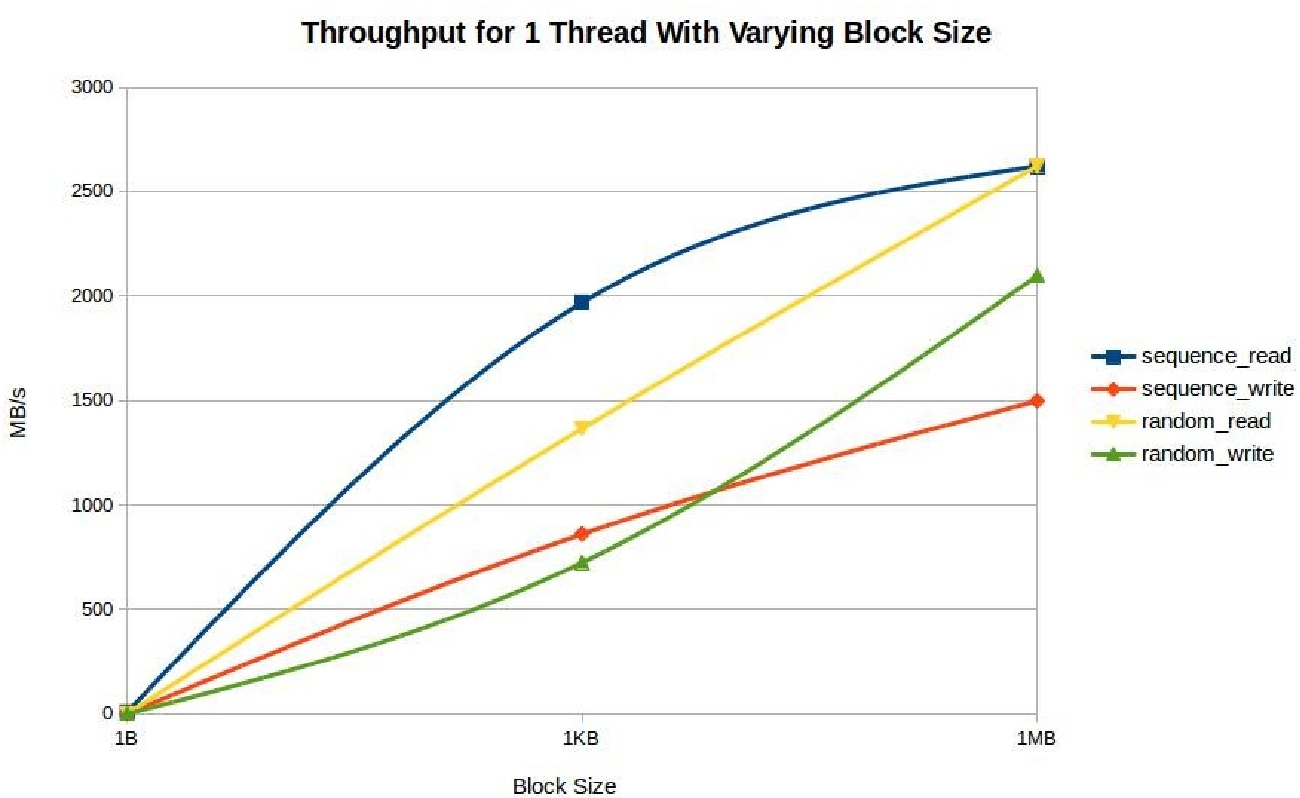
Random One Thread

Block Size	Read	Write
1B	2.381	0.751
1KB	1365.33	724.637
1MB	2621.44	2097.152

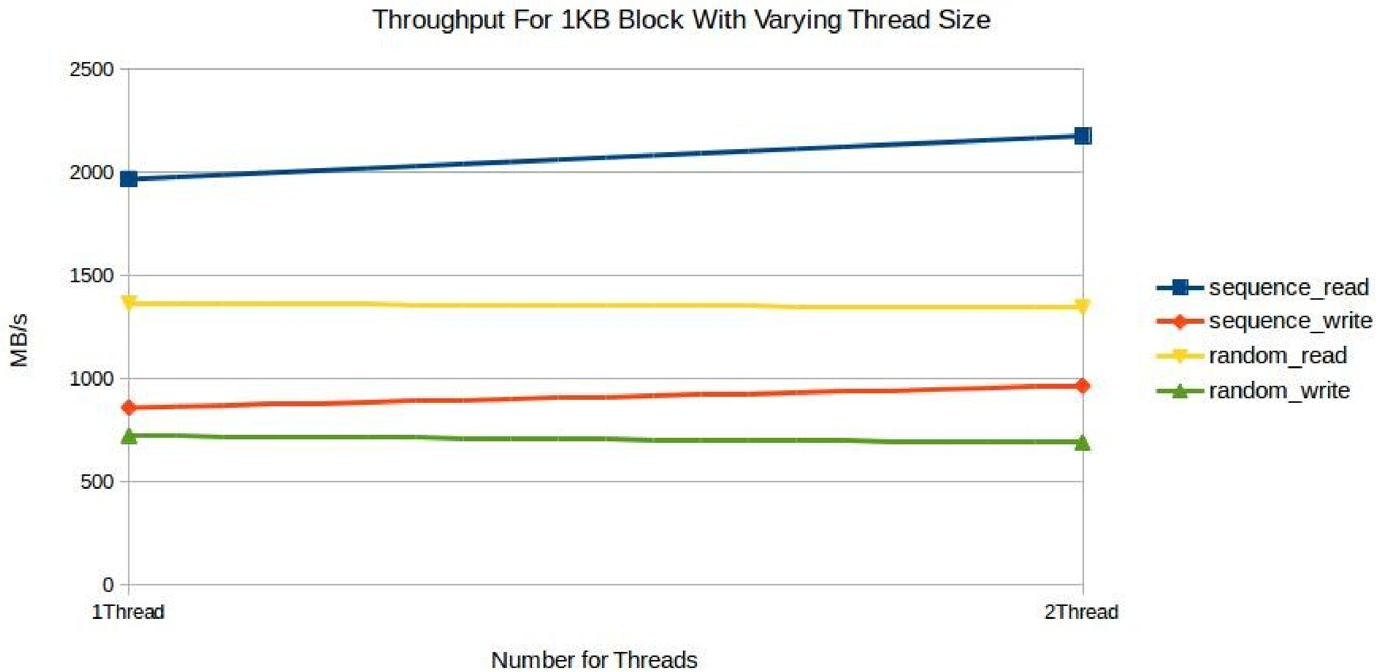
Random Two Thread

Block Size	Read	Write
1B	2.325	0.719
1KB	1347.36	691.89
1MB	1048.57	1747.62

Throughput Graph for One Thread



Throughput Graph for 1KB Block



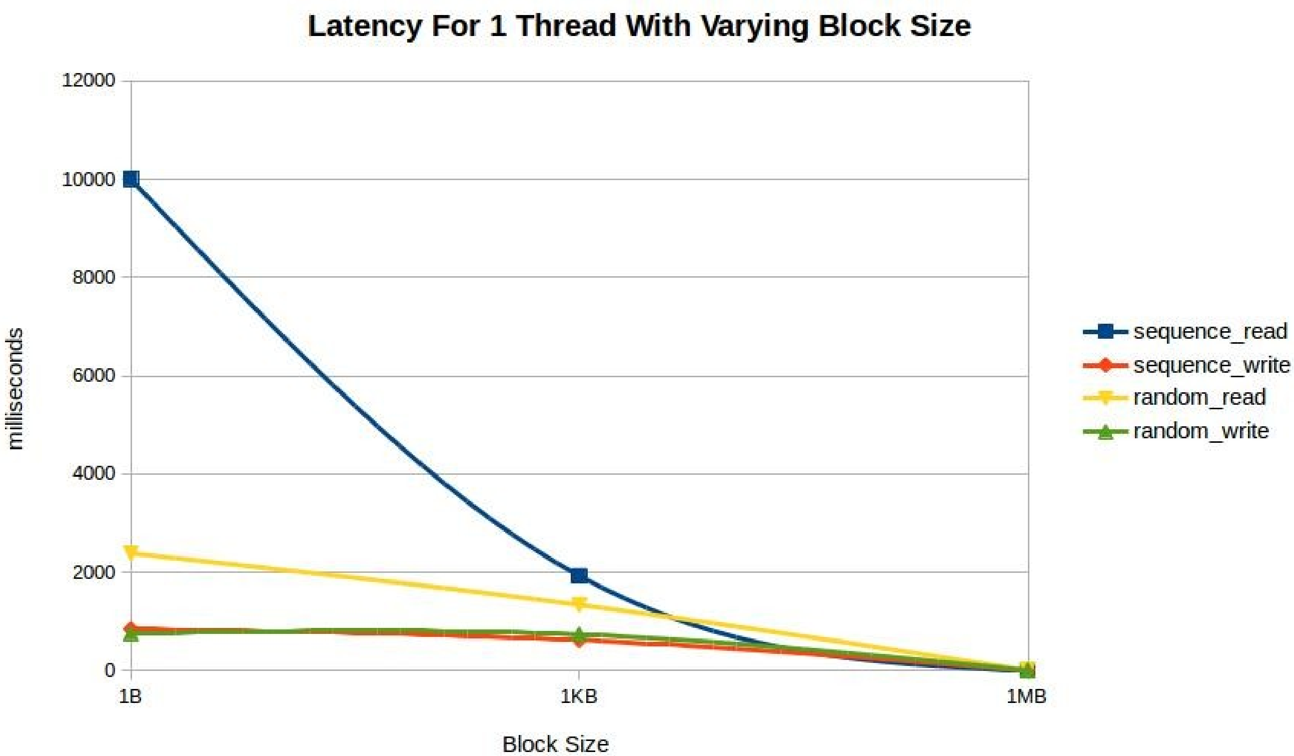
Sequential One Thread

Block Size	*Read*	*Write*
1B	10000	840.33
1KB	1923.07	617.28
1MB	2.5	1.4285

Random One Thread

Block Size	*Read*	*Write*
1B	2380.95	751.87
1KB	1333.33	729.93
1MB	2.5	2

Latency Graph for One Thread



IOzone utility measures:

Below are the value which I got by using the Iozone utility tool on Amazon EC2.

```
[ec2-user@ip-172-31-19-74 current]$ ./iozone -a -i 0 -i 1 -i 2 -s 1024
        Iozone: Performance Test of File I/O
                Version $Revision: 3.394 $
                Compiled for 64 bit mode.
                Build: linux

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer.
                     Ben England.

        Run began: Fri Feb 12 21:09:24 2016

        Auto Mode
        File size set to 1024 KB
        Command line used: ./iozone -a -i 0 -i 1 -i 2 -s 1024
        Output is in Kbytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 Kbytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
                                                              random  random    bkwd   record   stride
              KB  reclen   write rewrite    read    reread    read   write    read  rewrite     read   fwrite frewrite   fread  freread
            1024       4 1741104 4470172 11520658 10769573 9765601 4356809
            1024       8 2106609 5065980 12208350 1597796213305116 5251818
            1024      16 2178189 4995276 14871477 1529515713472053 5751117
            1024      32 2192645 4433259 13472053 1461839312790037 5950309
            1024      64 2222141 5751117 14618393 1461839313472053 6244448
            1024     128 2231377 4995276 12639479 1190382312037272 6244448
            1024     256 2305644 5226256 11368190 1190382311773301 5536137
            1024     512 2326879 5417427 11645609 1177330111398360 5303701
            1024    1024 2323103 3436508 11741116 1190382311249091 5751117

iozone test complete.
```

Below table shows the comparison between IOzone utility and the values that I got using my Disk benchmarking program. Values are for 1MB block running on 1 thread (All readings are in MB/s).

Functions	*IOzone values*	*My program values*
Sequential Read	11520.66	2621.44
Sequential Write	1741.10	1497.96
Random Read	9765.60	2621.44
Random Write	4356.80	2097.152

Theoretical peek performance of the disk

```
[ec2-user@ip-172-31-19-74 ~]$ sudo hdparm -tT /dev/sda

/dev/sda:
 Timing cached reads:   20704 MB in  2.00 seconds = 10361.80 MB/sec
 Timing buffered disk reads: 244 MB in  3.00 seconds =  81.29 MB/sec
[ec2-user@ip-172-31-19-74 ~]$
```

Efficiency of Iozone to Theoretical: (9765.60/10361)*100 = 94.25%

Result Interpretation:

From the tables and graphs it is clear that the disk performance increases with the increase in concurrency and increase in block size. We can also notice that sequential access is faster than the random access. This is obvious as random access increases the number of overheads like seeking the current pointer to the specific right position which might be physically far from the its old position.

Thus limitations of traditional hard disc leads to overheads during random access.

Throughput of a disk is **Average size(Block Size) * Operations per second** which is measured in mb/s.

Latency is the time taken to complete the input output request and is measured in milliseconds. It is also inversely proportional to the Throughput.

Issues Encountered:

A file opening in the cache was the biggest problem which would make the read and write throughput to shoot to very high values. The solution to this was to increase the number of iterations for the read and write operation forcing the cache to get filled up and then result into a cache miss. Programatically, as I was doing the benchmarking using Java, I was getting hard time to randomly access the file and write or read using the traditional classes. Needed to do some background study on “RandomAccessFile” class and its methods to get the task done.

4 MEMORY BENCHMARKING

For Memory Benchmarking sequential and random access of memory are performed for varying block sizes of 1B, 1KB and 1MB and varying number of threads (1 or 2). Below are the results of the experiments that I conducted.

Memory Performance: Throughput

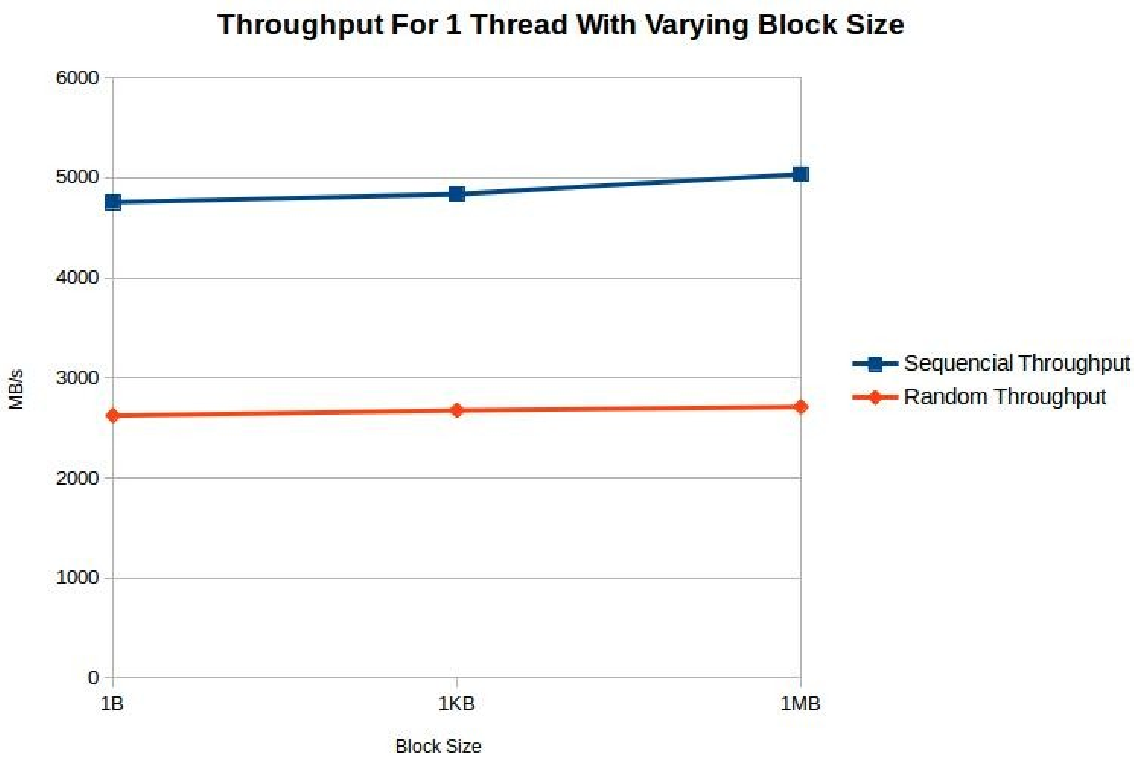
Sequential

Block_Size	*Threads*	*Throughput*
1B	1	4758.3
1B	2	4484.3
1KB	1	4837.12
1KB	2	4524.89
1MB	1	5037.12
1MB	2	4610.34

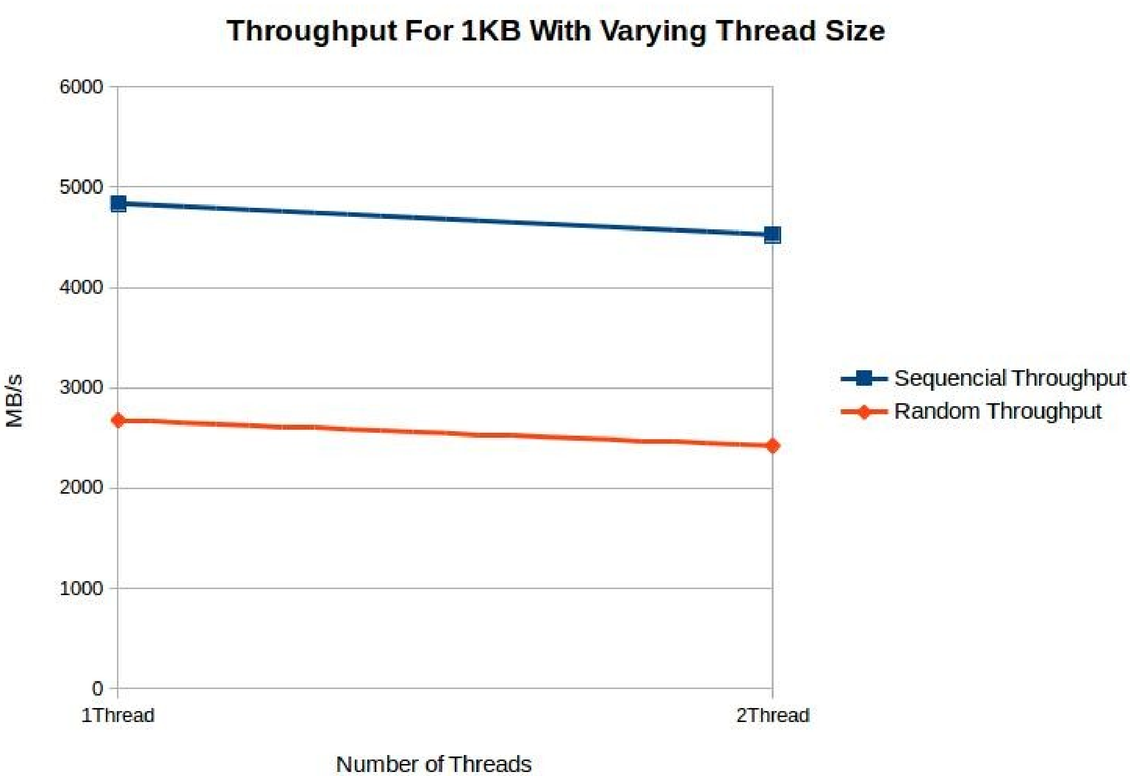
Random

Block_Size	*Threads*	*Throughput*
1B	1	2611.21
1B	2	2403.846
1KB	1	2674.312
1KB	2	2421.307
1MB	1	2710.312
1MB	2	2457.002

Throughput Graph for One Thread



Throughput Graph for 1KB Block



Disk Performance: Latency

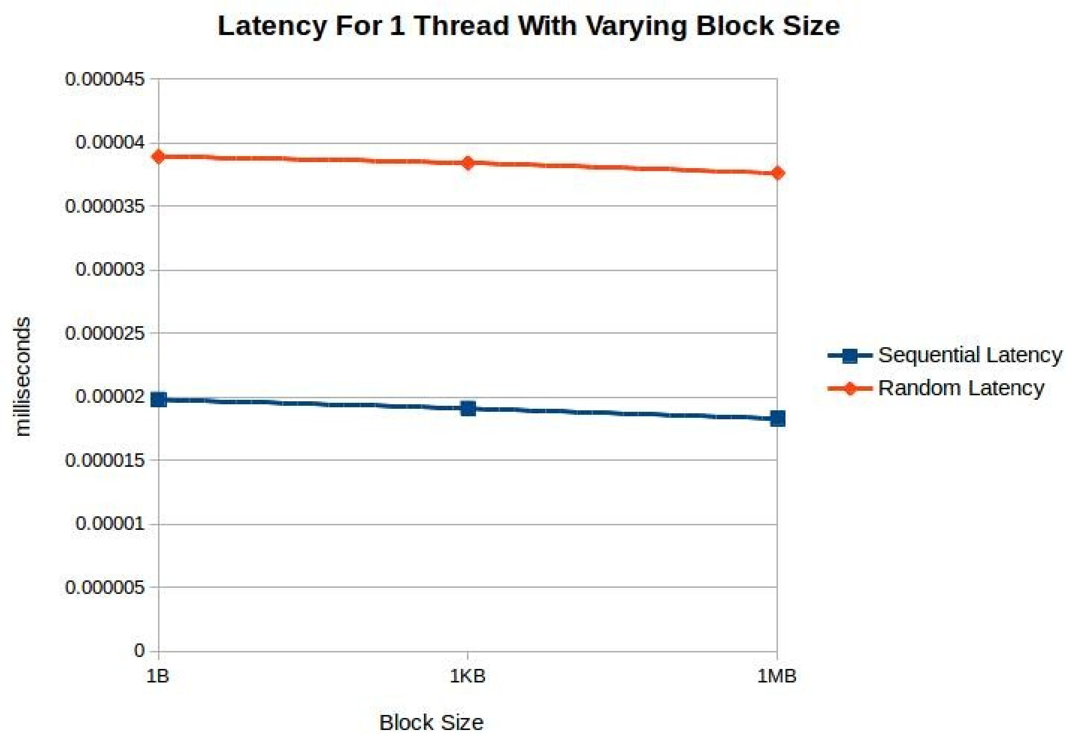
Sequential

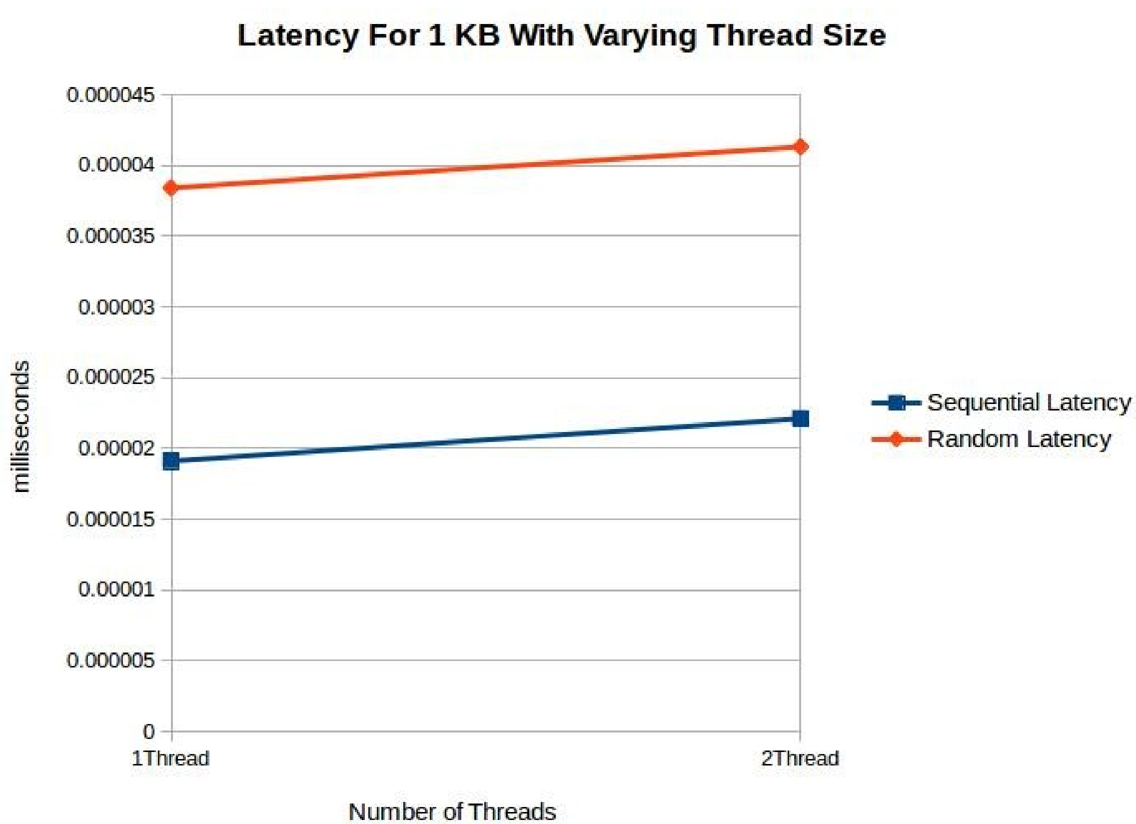
Block_Size	*Threads*	*Latency*
1B	1	0.0000198
1B	2	0.0000223
1KB	1	0.0000191
1KB	2	0.0000221
1MB	1	0.0000183
1MB	2	0.0000212

Random

Block_Size	*Threads*	*Latency*
1B	1	0.0000389
1B	2	0.0000416
1KB	1	0.0000384
1KB	2	0.0000413
1MB	1	0.0000376
1MB	2	0.0000407

Latency Graph for One Thread





Stream Benchmark utility measures:

Below are the value which I got by using the Stream Benchmark utility tool on Amazon EC2.

```
[ec2-user@ip-172-31-19-74 ~]$ gcc stream.c
[ec2-user@ip-172-31-19-74 ~]$ ./a.out
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 28514 microseconds.
   (= 28514 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            5436.1     0.030230     0.029433     0.032319
Scale:           5371.8     0.030481     0.029785     0.032587
Add:             7682.5     0.032273     0.031240     0.034070
Triad:           7247.5     0.034454     0.033115     0.036203
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
[ec2-user@ip-172-31-19-74 ~]$
```

Theoretical Throughput = 10361.80 MB/s, Latency = 0.27ms

For my results,

the optimal block size is 1MB for a single thread.

Practical Throughput = 5037.12 MB/s and Latency = 0.0000183 ms

Efficiency = (5037.12/10361.8)*100 = 48.61%

For Stream Benchmark utility tool,

Practical Throughput = 6434.47 MB/s and Latency = 0.0000318 ms

Efficiency = (6434.47/10361.8)*100 = 62.09%

Result Interpretation:

From the tables and graphs iwe can observe that as we increase the number of block size Sequential and Random throughput also increases and Latency decreases.

If we increase the number of threads then Sequential and Random throughput decreases and latency. This is because of the overheads associated with threads.

For all experiments the Sequential throughput would be significantly higher than Random throughput and visa versa for Latency.